First Hit

Generate Collection Print

L9: Entry 1 of 3 File: PGPB Aug 1, 2002

DOCUMENT-IDENTIFIER: US 20020103819 A1
TITLE: Technique for stabilizing data in a non-log based information storage and retrieval system

Brief Description of Drawings Paragraph:
[0031] FIG. 7B shows a block diagram of a specific embodiment of a database server 700 which may be used for implementing the information storage and retrieval technique of the present invention.

Detail Description Paragraph:
[0095] FIG. 7A shows a block diagram of a specific embodiment of a client library 750 which may be used in implementing the information storage and retrieval technique of the present invention. As shown in FIG. 7A, the client library 750 includes a database (DB) library portion 780 which provides a mechanism for communicating with a database server of the present invention such as that shown, for example, in FIG. 7B.

Detail Description Paragraph:
[0097] Further, as shown in the embodiment of FIG. 7A, the client library includes an object workspace 762 which may be used for caching objects for fast access. The client library may also include a schema manager 768 for handling schema modifications and for validating updates against the application schema. The RPC layer 764 and network layer 766 may be used to control the connections to the database server and to control the transfer of information between the client and server.

Detail Description Paragraph:
[0098] FIG. 7B shows a block diagram of a specific embodiment of a database server 700 which may be used in implementing the information storage and retrieval technique of the present invention. According to at least one embodiment, the database server 700 may be configured as an object server, which receives and processes object updates from clients and also delivers requested objects to the clients.

Detail Description Paragraph:
[0099] As shown in FIG. 7B, the database server includes an Object Manager 702 for managing objects stored in the database. In performing its functions, the Object Manager may rely on internal structures, such as, for example, B-trees, sorted lists, large objects (e.g. objects which span more than one disk page), etc. According to a specific embodiment, Object Manager 702 may be responsible for creating and/or managing user objects, user indexes, etc. The Object Manager may make calls to the other database managers in order to perform specific management functions. The Object Manager may also be responsible for managing conversions between user visible objects and internal database objects.

Detail Description Paragraph:
[0100] The database server may also include an Object Table Manager 706, which may be responsible for managing Object Table entries, including object entries in both the Memory Object Table portion and Persistent Object Table portion of the Object Table.

Detail Description Paragraph:
[0101] The database server may also include a Version Collection (VC) Manager 703, which may be responsible for managing version collection details such as, for example, clearing obsolete data, compaction of non-obsolete data, cleaning up Object Table data, etc. According to one implementation, both the VC manager and the Object Manager may call upon the Object Table Manager for performing specific operations on data stored in the Object Table.

Detail Description Paragraph:
[0102] The database server may also include a Transaction Manager 704, which may be responsible for managing transaction operations such as, for example, committing transactions, stalling transactions, aborting transactions, etc. According to a specific implementation, a transaction may be defined as an atomic update of a portion of data in the database. The Transaction Manager may also be responsible for managing serialized and consistent updates of the database, as well as managing atomic transactions to help insure recovery of the database in the event of a software or disk crash.

Detail Description Paragraph:
[0103] The database server may also include a Cache Manager 708, which may be responsible for managing virtual memory operations. This may include managing where specific data is to be stored in the virtual memory (e.g. either on disk or in the data server cache). According to a specific implementation, the Cache Manager may communicate with the Disk Manager 710 for accessing data in the persistent memory. The Cache Manager and Disk Manager may work together to ensure parallel reads and writes of the data across multiple disks 740. The Disk Manager 710 may be responsible for disk I/O operations, and may also be responsible for load balancing operations between multiple disks 740 or other persistent memory devices.

Detail Description Paragraph:
[0104] The database server 700 may also include an SQL execution engine 709 which may be configured to process SQL requests directly at the database server, and to return the desired results to the requesting client.

Detail Description Paragraph:
[0105] The database server 700 may also include a Version Manager 711 which may be responsible for providing consistent, non-blocking read access to the database data at anytime, even during updates of the database data. This feature is made possible by the intrinsic versioning architecture of the database server of the present invention.

Detail Description Paragraph:
[0106] If desired, the database server 700 may also include a Checkpoint Manager 712 which may be responsible for managing checkpointing operations performed on data within the database. According to a specific embodiment, the VC Manager 704 and Checkpoint Manager 712 may work together to automatically reclaim the disk space used by obsolete versions of objects that have been deleted. The Checkpoint Manager may also be responsible for handling the checkpoint mechanism that identifies the stable data in the persistent memory 740. This helps to guarantee a fast restart of the database server after a crash, which, according to at least one embodiment, may be independent of the amount of data stored in the database.

Detail Description Paragraph:
[0107] As described previously, the database server 700 includes an Object Table 720 which provides a mapping between the logical object identifiers (OIDs) and the physical address of the objects stored in the database.

Detail Description Paragraph:
[0108] It will be appreciated that alternate embodiments of the database server and

client library of the present invention may not include all the elements and/or features described in the embodiments of FIGS. 7A and 7B. The specific configurations of such alternate embodiments may vary depending upon the desired specifications, and will be readily apparent to one having ordinary skill in the art.

Detail Description Paragraph:
[0109] According to at least one embodiment, the database system of the present invention may be designed or configured as a client-server system, wherein applications built on top of a client database library talk with a database server using database Remote Procedure Calls (RPCs). A database client implemented on a client device may exchange objects with the database server. In one implementation, objects which are accessed through the client library may be cached in the client workspace for fast access. Moreover, according to one implementation, only the essential or desired portions of the data pages are provided by the database server to the client. Unnecessary data such as, for example, index pages, internal structures, etc., are not sent to the client machine unless specifically requested. Additionally, it will be appreciated that the information storage and retrieval technique of the present invention differs greatly from that of conventional RDBMS techniques which only return a projection back to the client rather than objects which can be modified directly by the client workspace.

Detail Description Paragraph:
[0110] Additionally, according to a specific embodiment, the database server of the present invention may be implemented on top of kernel threads, and may be configured to scale linearly as new CPUs or new persistent memory devices (e.g. disks) are added to the system.

Detail Description Paragraph:
[0112] Further, unlike conventional RDBMS systems which use transaction log file techniques to ensure database integrity, the information storage and retrieval system of the present invention may be configured to achieve database integrity without relying upon transaction logs or conventional transaction log file techniques. More specifically, according to a specific implementation, the database server of the present invention is able to maintain database integrity without performing any transaction log activity. Moreover, the intrinsic versioning feature of the present invention may be used to ensure database recovery without incurring overhead due to log transaction operations.

Detail Description Paragraph:
[0113] According to one embodiment, intrinsic versioning is the automatic generation and control of object versions. According to traditional database techniques, when changes or updates are to be performed upon objects stored in a conventional database, the updated data must be written over the old object data at the same physical location in the database which has been allocated for that particular object. This feature may be referred to as positional updating. In contrast, using the technique of the present invention, when data relating to a particular object has been changed or modified, a copy of the new object version may be created and stored in the database as a separate object version, which may be located at a different disk location than that of any previously saved versions of the same object. In this way, the database system of the present invention provides a mechanism for implementing non-positional data updates.

Detail Description Paragraph:
[0117] According to a specific embodiment, the database server of the present invention may be configured as a general purpose object manager, which operates as a back-end server that manages a repository of persistent objects. Client applications may connect to the server through a data network or through a local transport. The database server of the present invention may be configured to ensure that all that objects stored therein remain available in a consistent state, even

in the presence of system failures. Additionally, when server clients access a shared set of objects simultaneously in a read or write mode, the database server of the present invention may be configured to ensure that each server client gets a consistent view of the database objects.

Detail Description Paragraph:
[0127] FIG. 10 shows a flow diagram of a Cache Manager Flush Procedure 1000 in accordance with a specific embodiment of the present invention. According to a specific implementation, the Cache Management Flush Procedure 1000 may be configured as a process in the database server which runs asynchronously from other processes such as, for example, the Disk Manager Flush Procedure 1100 of FIG. 11.

Detail Description Paragraph:
[0129] FIG. 11 shows a flow diagram of a Disk Manager Flush Procedure 1100 in accordance with a specific embodiment of the present invention. According to one embodiment, a separate thread or process of the Disk Manager Flush Procedure may be implemented at each respective writer thread (e.g. 920A, 920B, 920C, etc.) running on the database server. Further, according to at least one embodiment, each writer thread may be configured to write to a designated disk or persistent memory device of the persistent memory. For purposes of illustration, it will be assumed that the Disk Manager Flush Procedure 1100 is being implemented at the Writer Thread A 920A of FIG. 9A.

Detail Description Paragraph:
[0141] Unlike conventional database recovery techniques, the technique of the present invention does not use a transaction log file to provide database recovery functionality. Further, as explained in greater detail below, the amount of time it takes to fully recover the database information using the technique of the present invention may be independent of the size of the database.

Detail Description Paragraph:
[0150] According to a specific embodiment, once a Commit Transaction object has been flushed to the persistent memory, all updates associated with the Transaction ID of the Commit Transaction object may be considered to be stable for the purpose of rebuilding the database. Thus, it will be appreciated that, according to one embodiment, database recovery may be performed without the use of a transaction log file. Further, since the data associated with a given committed transaction is capable of being recovered once the transaction has been committed, database recovery may be performed without performing any checkpointing of the committed transaction or related data.

Detail Description Paragraph:
[0152] Initially, upon restart or initialization of the database server, each of the disks in the database persistent memory may be scanned in order to determine (1402) whether all of the disks are stable. According to one implementation, the header portion of each disk may be checked in order to determine whether the disk had crashed or was gracefully shut down. According to the embodiment of FIG. 14, if a disk was gracefully shut down, then the disk is considered to be stable.

Detail Description Paragraph:
[0153] If it is determined that all database disks are stable, then it may be assumed that all data in each of the disks is stable. Accordingly, a Graceful Restart Procedure may then be implemented (1404). During the Graceful Restart Procedure, the memory portion of the Object Table (i.e., Memory Object Table) may be created by loading into the program memory information from the portion of the Object Table that has been stored in the persistent memory (i.e., the Persistent Object Table). Thereafter, the database server may resume its normal operation.

Detail Description Paragraph:
[0155] FIG. 15 shows a flow diagram of a Crash Recovery Procedure 1500 in

accordance with a specific embodiment of the present invention. According to a specific embodiment, the Crash Recovery Procedure 1500 may be used to rebuild or reconstruct the Object Table using the data stored in the persistent memory. In on implementation, the Crash Recovery Procedure 1500 may be implemented, for example, by the Object Manager following a crash or failure of the database server.

Detail Description Paragraph:
[0162] It will be appreciated that since the Crash Recovery Procedure of FIG. 15 involves at least one scan of the entire data set, full recovery of a relatively large database may be quite time consuming. In order to reduce the recovery time needed for rebuilding the database following a system crash, an alternate embodiment of the present invention provides a database recovery technique which utilizes a checkpointing mechanism for creating stable areas of data in the persistent memory which may immediately be recovered upon restart.

Detail Description Paragraph:
[0169] According to a specific implementation, the stable Persistent Object Table 1714 and stable data 1716 represent information which has been stored in the persistent memory using the checkpointing mechanism of the present invention. As explained in greater detail with respect to FIGS. 16A and 16B, database recovery may be achieved by retrieving the stable Persistent Object Table 1714 and using the unstable data 1722 to patch data retrieved from the stable Persistent Object Table to thereby generate a recovered, stable Object Table.

Detail Description Paragraph:
[0170] FIG. 16A shows a flow diagram of a Checkpointing Restart Procedure 1600 in accordance with a specific embodiment of the present invention. The Checkpointing Restart Procedure 1600 may be implemented, for example, by the Object Manager following a restart of the database system. For purposes of illustration, it is assumed that the Checkpointing Restart Procedure 1600 is being implemented on a database server system which includes a persistent memory storage device as illustrated in FIG. 17 of the drawings.

Detail Description Paragraph:
[0174] One advantage of the checkpointing mechanism of the present invention is that it provides for improved crash recovery performance. For example, since the stable data in the database may be quickly and easily identified by accessing the Allocation Map 1706, the speed at which database recovery may be achieved is significantly improved. Further, at least a portion of the improved recovery performance may be attributable to the fact that the stable data does not have to be analyzed to rebuild the post recovery Object Table since this information is already stored in the stable Object Table 1714. Thus, according to a specific embodiment, only the unstable data identified in the persistent memory need be analyzed for rebuilding the remainder of the post recovery Object Table.

Detail Description Paragraph:
[0232] For example, according to one embodiment of the present invention, disk Allocation Maps are not stored on their respective disks (or other persistent memory devices), but rather are stored in volatile memory such as, for example, the data server cache. According to this embodiment, when a particular disk page of the persistent memory is to be freed, the Free Flag may be set in the Allocation Map entry corresponding to that disk page, and a blank page written to the physical location of the persistent memory which had been allocated for that particular disk page. According to one implementation, the blank page data may be written to the persistent memory in order to assure proper data recovery in the event of a system crash. For example, if a systems crash were to occur, the Allocation Map stored in the data server cache would be lost. Therefore, recovery of the database would need to be achieved by scanning the persistent memory for data in order to rebuild the Allocation Map. The blank pages written to the persistent memory ensure that obsolete or stale data is not erroneously recovered as valid data.

Detail Description Paragraph:
[0241] Referring now to FIG. 26, a network device 10 suitable for implementing the information storage and retrieval technique of the present invention includes at least one central processing unit (CPU) 61, at least one interface 68, memory 62, and at least one bus 15 (e.g., a PCI bus). When acting under the control of appropriate software or firmware, the CPU 61 may be responsible for implementing specific functions associated with the functions of a desired network device. When configured as a database server, the CPU 61 may be responsible for such tasks as, for example, managing internal data structures and data, managing atomic transaction updates, managing memory cache operations, performing checkpointing and version collection functions, maintaining database integrity, responding to database queries, etc. The CPU 61 preferably accomplishes all these functions under the control of software, including an operating system (e.g. Windows NT, SUN SOLARIS, LINUX, HPUX, IBM RS 6000, etc.), and any appropriate applications software.

Detail Description Paragraph:
[0243] According to at least one embodiment, the network device 10 may also include persistent or non-volatile memory 74. Examples of non-volatile memory include hard disks, floppy disks, magnetic tape, optical media such as CD-ROM disks, magneto-optical media such as floptical disks, etc.

Detail Description Paragraph:
[0247] Because such information and program instructions may be employed to implement the systems/methods described herein, the present invention relates to machine readable media that include program instructions, state information, etc. for performing various operations described herein. Examples of machine-readable media include, but are not limited to, magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and hardware devices that are specially configured to store and perform program instructions, such as read-only memory devices (ROM) and random access memory (RAM). The invention may also be embodied in a carrier wave travelling over an appropriate medium such as airwaves, optical lines, electric lines, etc. Examples of program instructions include both machine code, such as produced by a compiler, and files containing higher level code that may be executed by the computer using an interpreter.